ELC-Europe 2016

Thursday, October 13 • 11:15 - 12:05

No, It's Never Too Late to Upstream Your Legacy Linux Based Platform

Agenda

- Questions and Answers
  - Why Upstreaming ?
  - Why mainline kernel ?
  - Why push code ?
  - Which workflow ?
  - How long does it take ?
  - How hard it is for an “old” platform ?
  - What about the benefits ?

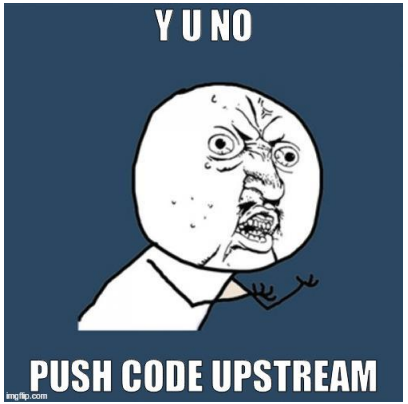
Question 1

Why should I push code for my (legacy) linux based platform ?

Why should I push code for my (legacy) linux based platform ?

Always the same question, always the same answers.

More and more vendors had understood the strategic advantages to push code upstream.



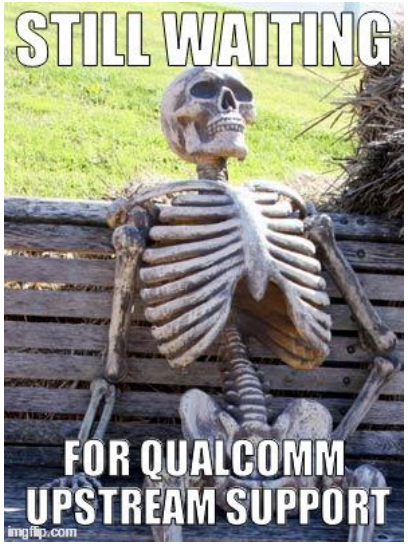
But still very large vendors does not understand :

- They should publish the kernel code (sigh)
- They should push their kernel code in a git repo
- They should push clean code
- They should rework and push code upstream
- They should have a upstream based workflow for their linux products

Why should I push code for my (legacy) linux based platform ?

Hopefully, we can count some vendors that really participate in the upstream work like :

- Intel
- IBM
- Texas Instruments
- Atmel (Microchip)
- Broadcom
- Renesas
- Freescale (NXP)
- ...



Why should I push code for my (legacy) linux based platform ?

This is for large corporations, what about smaller vendors ?

We can them separate in two :

- SoC vendors
- Boards makers (or ODMs)

For SoC vendors, Semiconductor World is a tough, and design costs for a single SoC are really high, then IP companies like ARM, Synopsys, Cadence… takes a big chunk of the costs and sometimes won't let you use their code for public work.

Why should I push code for my (legacy) linux based platform ?

For Board makers, the situation is even more complex.

Board makers are provided (when possible) with a very custom and complex BSP or SDK which targets either all possible uses cases of the SoC and very specific Reference Design uses cases.

Yes, sometimes ODMs does not even provide the kernel source (sigh) and even nothing when Android is involved in the product, because the Board vendor will only need to develop its custom Android App.

Why should I push code for my (legacy) linux based platform ?

But some SoC vendors participate actively to make (*part*) of their SoC family work (*partly*) upstream.

But these BSPs are often kept on old kernel releases (almost 3.x).

Some vendors even synchronize regularly with the latest Stable kernel and rebase their BSP on it (Renesas, …)

Open Source strategy of each SoC vendor should become the priority in the SoC selection for a new product design.

Question 2

Hmm, why Upstream kernel is so important after all ?

Hmm, why Upstream kernel is so important after all ?

Relying on Upstream kernel is important because (at least):

- You can get new features
  - Network features is the most important
  - USB Drivers
  - Performance Enhancement
  - …
- You can have bugfixes
- You can have enhanced security features
- You can the improve stability of your product

Question 3

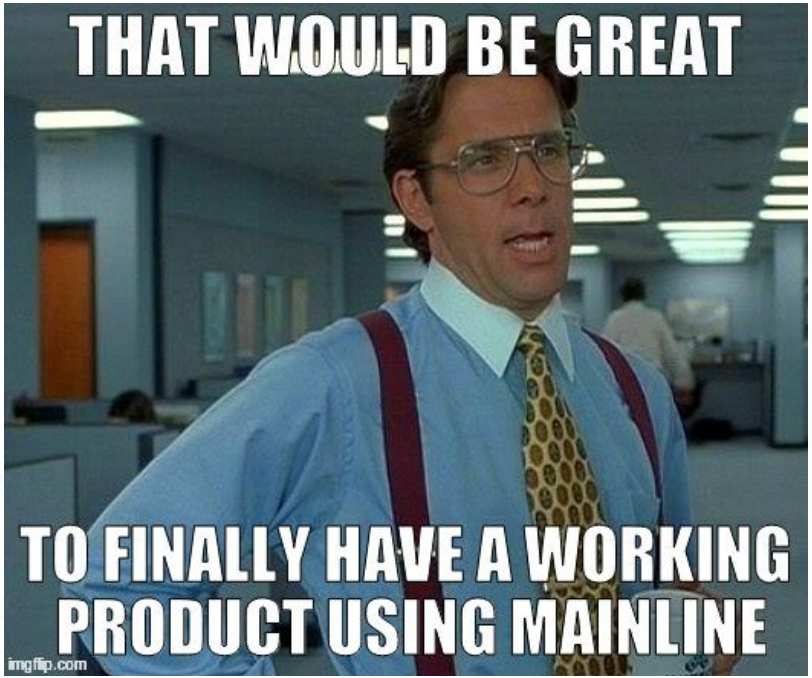
Making a product work is complex enough, why push code upstream ?

Making a product work is complex enough, why push code upstream ?

For two simple reasons :

- Code maintenance ease
- Fair return to the community
- Strengthen the Linux Platform

It’s all !



Making a product work is complex enough, why push code upstream ?

- Code maintenance ease ?

be part of the kernel evolution !
code will stay clean and get API changes !

Basic Board/SoC support will help following each linux releases.

In this case, when a recent kernel is needed, forward porting will be faster and can be done regularly.

→ Gain time, gain money !

Making a product work is complex enough, why push code upstream ?

- Fair return to the community

This seems to be a communist concept !

But, if you stay realistic, Linux is free, Linux is powerful, modern, modular, clean (as possible) and can provide an industrial grade Operating System support.

Did you try to find a non-free alternative ? QNX ? Windows NT ?

These alternative are expensive, very expensive and often requires huge royalties !

Making a product work is complex enough, why push code upstream ?

- Strengthen the Linux Platform

Large number of contributors and great amount of very different platform to support are the keys to strenghten Linux, stabilize the code, enhance portability and ensure longevity of the project.

Don’t worry, all SoC designs are different, and it’s good !

Question 7

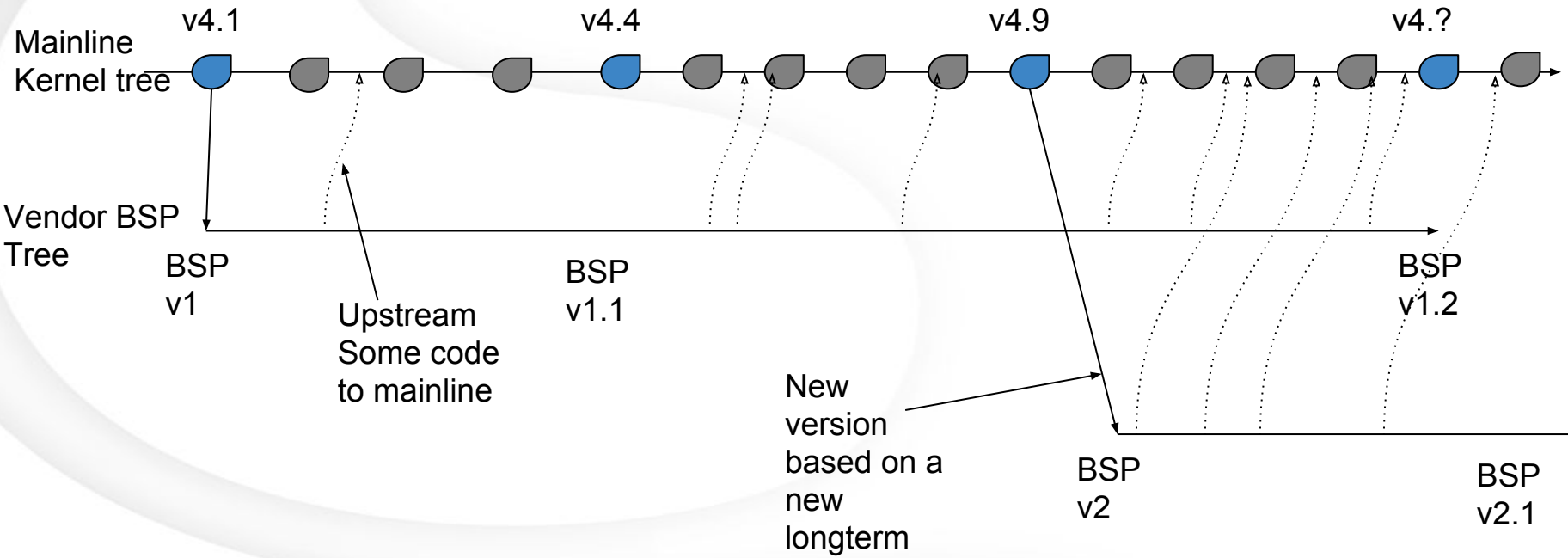
How should I organize my upstream workflow ?

How should I organize my upstream workflow ?

Some workflow examples :

- Maintain separate trees, one for the BSP and upstream to mainline for client that require recent linux versions
- Port all code from a stable BSP, when nearly complete rebase the BSP on the new linux version
- Iterate by porting the BSP kernel on each long term version, and in the meanwhile upstream as much as possible, …

How should I organize my upstream workflow ?



How should I organize my upstream workflow ?

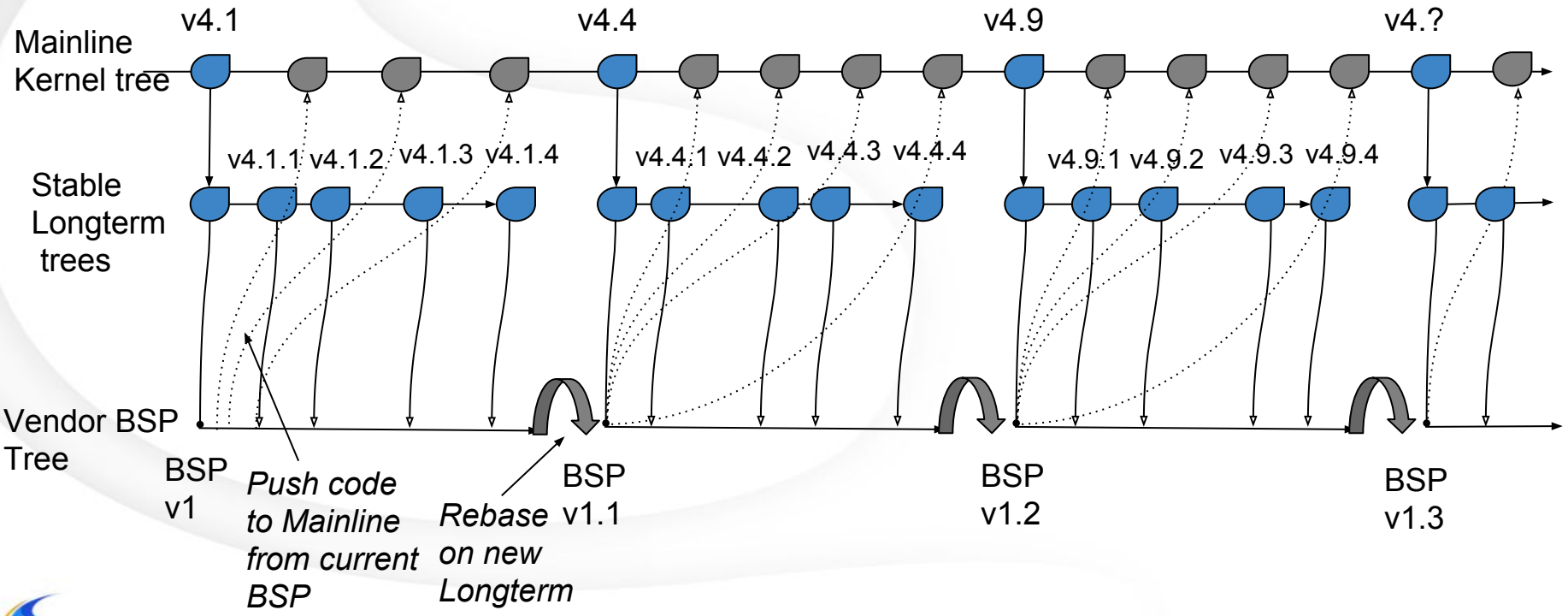
Pros :

- BSP is stable
- Mainline kernel will sometime good support
- Some clients can use mainline

Cons :

- BSP kernel version becomes old at some time
- Rebasing on a new longterm will need a lot of work
- Back porting bugs and security fixes will become harder
- Back porting new features will create a hard to maintain kernel

How should I organize my upstream workflow ?



How should I organize my upstream workflow ?

Pros :

- BSP has always longterm new features, ~each year
- Mainline kernel will get near ~100% support
- BSP driver are reworked and cleaned up

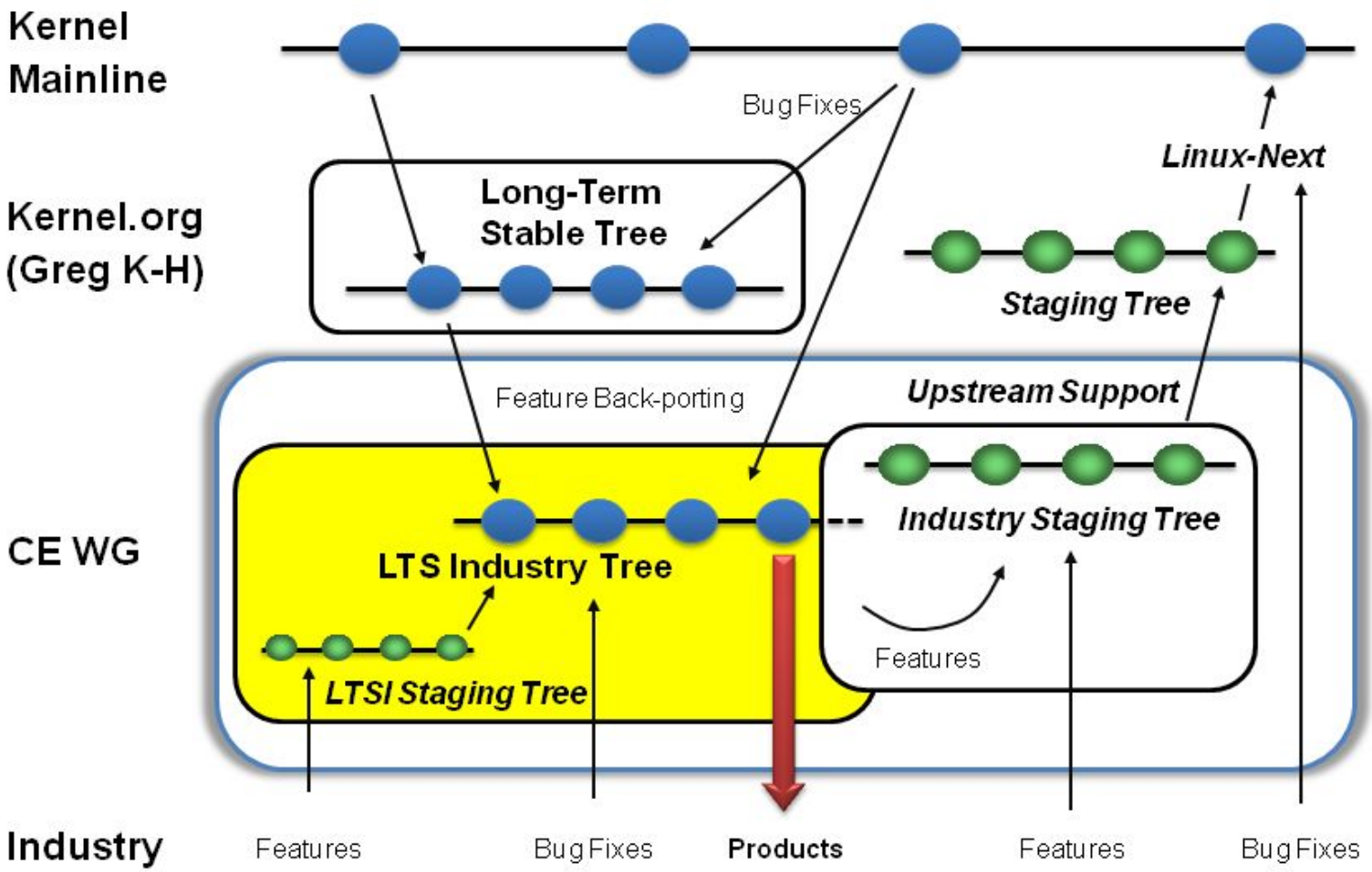
Cons :

- Must maintain a separate team for upstreaming
- Rebasing on each longterm can be complex since drivers has changed
- 1y BSP update may be short for long term supported products, old BSP versions should also have bug and security fixes

How should I organize my upstream workflow ?

LTSI

LTSI initiative offers a way for vendors to base their product release on a stable kernel following the annual long-term version selection.


Kernel
Mainline
Bug Fixes
Linux-Next
Kernel.org
(Greg K-H)
Long-Term
Stable Tree
Staging Tree
Feature Back-porting
Upstream Support
CE WG
Industry Staging Tree
LTS Industry Tree
Features
LTSI Staging Tree
Industry
Features
Bug Fixes
Products
Features
Bug Fixes

Question 5

How long will it take me to push all this code upstream ?

How much time will it take me to push all this code upstream ?

This a complex question without any clear answers.

The Linux development cycle must be understood !

Rework/Refactor is mandatory before and while submission retries.

Depending on subsystems and complexity, submission time can take most of the time.

How much time will it take me to push all this code upstream ?

The general mainlining workflow is to push code in each linux subsystems, one by one.

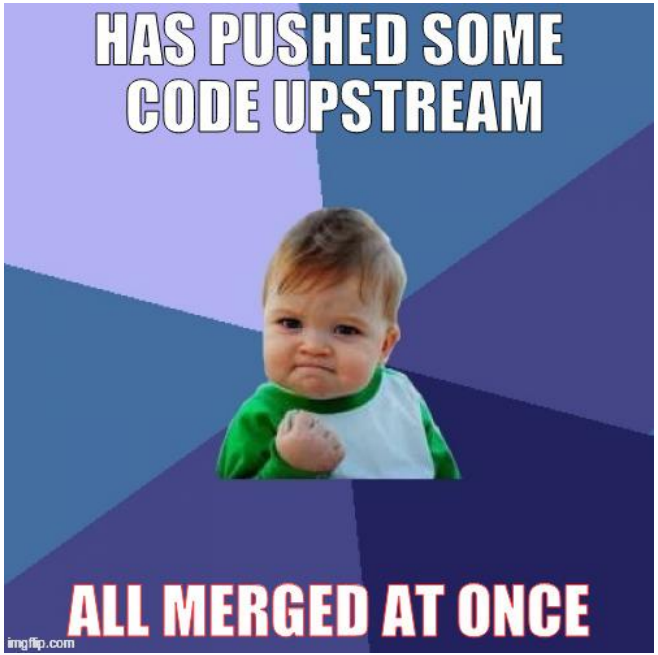
Coherency of the support for a platform is done over the time.

There is no current “methodology” to push an overall platform support at once, each maintainers will want to have control and review the code to conform to their habits and ease their future work.

How much time will it take me to push all this code upstream ?

This is why it can take over 2 or 3 releases to get a complete set of patches upstream and have a working kernel on a platform.

The initial support of a platform is the most frustrating period since it will need the full patchset to boot correctly, but for some reasons one subsystem will fail to merge on time for some reasons.



How much time will it take me to push all this code upstream ?

Some of the reasons for the delays are generally :

- Code does not match the subsystem style / code design / architecture (use of deprecated API, …)
- Code depends on headers part of an higher level subsystem (for examples dt-bindings includes for Device Tree support)
- Code depends on partly merged framework API
- Patch is posted too late, too close to the next merge window

How much time will it take me to push all this code upstream ?

Here are some very approximate times :

- Push initial support of a SoC : 2 or 3 versions → ~6 months
- For a simple driver, like PWM, I2C bus, … :
  - 1 week refactoring / cleanup / patchset preparation
  - 1 day to 1 week for each repost depending on complexity
  - 1 or 2 days for testing before and after merge window
- For a more complex driver like DRM, SATA or Audio driver
  - Initial refactoring time can be much longer, up to 1 or 2 months
  - Time for repost depends on testing time and size of refactoring
  - Such driver upstreaming could be iterated over multiple versions

How much time will it take me to push all this code upstream ?

Global times estimations :

- For a simple headless SoC with any power management :
  - 6 months to 9 months
  - Full time for 1 person, multiple resources won't speed up but will permit more drivers submitted / refactored / tested per version
- For a very complex SoC with Video, Power Management, DSP Audio, modem, …
  - 18 months to multiple years depending of the original code quality and SoC complexity (Bus scaling, complex RPC code for Co-processors, complex Audio, secret GPU code, …)

Question 6

I have an old Linux port for my SoC, how hard it would be to upstream this ?

I have an old Linux port for my SoC, how hard it would be to upstream this ?

Since the Device Tree migration, the non-x86 support has changed a lot and simplified (is this the right word ?) support for complex SoCs by introducing some frameworks like :

- common clock framework
- pinctrl and gpiod
- generic interrupt
- reset
- drm
- ASoC
- ...

I have an old Linux port for my SoC, how hard it would be to upstream this ?

This means the traditional core in :

arch/arm/mach-mysoc/board.c
arch/arm/mach-mysoc/devices.c
arch/arm/mach-mysoc/pinmux.c
arch/arm/mach-mysoc/clock.c
arch/arm/mach-mysoc/include/mach/vmalloc.h
arch/arm/mach-mysoc/include/mach/mysoc.h
...

Is nearly finished !

I have an old Linux port for my SoC, how hard it would be to upstream this ?

With Device Tree support, the directory :

arch/arm/mach-mysoc/

Will only contain only very specific SoC code like SMP or special init code. And in ARM64, these directories are gone forever !

But where did the code go ?

I have an old Linux port for my SoC, how hard it would be to upstream this ?

In Device Tree, yes, but also in the linux subsystems directories like :

drivers/irqchip : for IRQ controller support
drivers/clocksource : for Tick and Clock source support
drivers/clk : For system clocks management
drivers/pinctrl : For pads, pins configuration and mux
drivers/reset : For internal reset lines control
…

All these coordinated by the device-tree nodes interconnections.

I have an old Linux port for my SoC, how hard it would be to upstream this ?

But, will it be hard ?

Yes

But you can have help, by using :

- Trainings (Linux Foundation, Free Electrons, …)
- Linux Experts (BayLibre, Free Electrons, Pengutronix, …)
- Working with the community as Greg KH explains

Question 7

What about the benefits ?

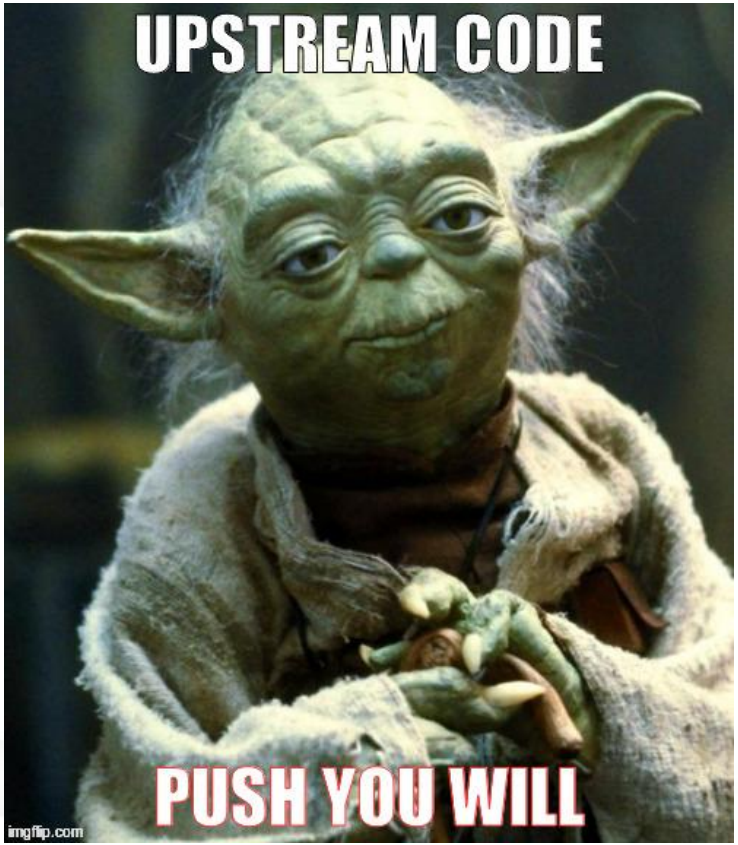
What about the benefits ?

- Increase in Code quality
  - External review can make the code better
- Minimize Code maintenance cost
  - At least for far less than off-tree
- Enable faster rebase and testing effort
- Clear and Open Software Strategy
  - No more obscure and non-reviewed dirty code
- Customer fidelity over well maintain codebase
- A good way to promote the company within the technical sphere
  - Could also make talented developers work for you

What about the benefits ?

- Money

Yes, it will minimize long term R&D costs !


UPSTREAM CODE
PUSH YOU WILL
imgflip.com

Ressources about mainlining

http://elinux.org/Kernel_Mainlining

- talks
- best practices

https://ltsi.linuxfoundation.org/what-is-ltsi/advantages-upstream-alignment

- “Marketing” Advantages